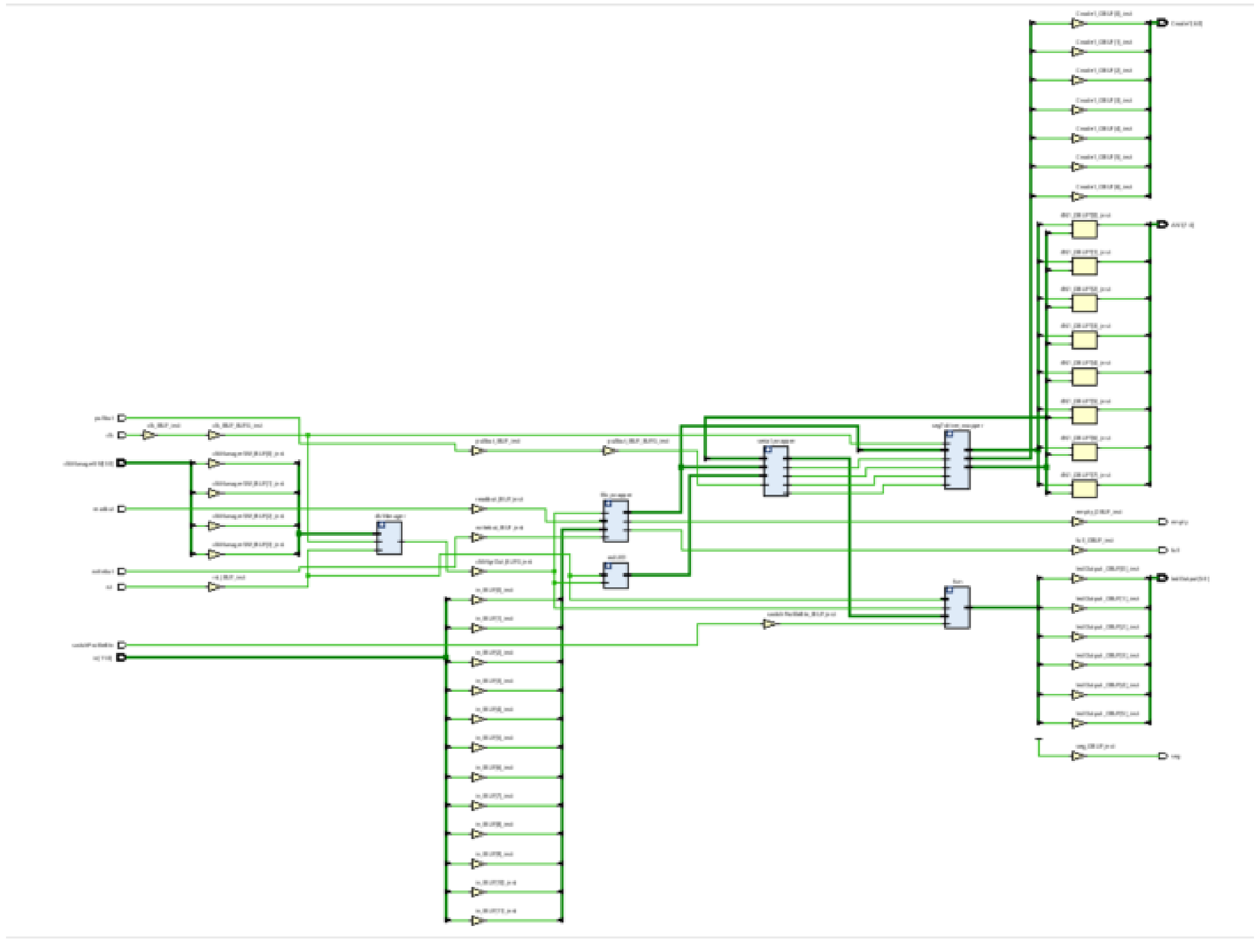
ECE3300L Lab10 Group H Report (Sherwin Sathish & Mohamed Hamida)

SCHEMATIC:



Xdc for top.v:

```
## This file is a general .xdc for the Nexys A7-100T
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project

## Clock signal
set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {clk}];


##Switches
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { redSW[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { redSW[1] }]; #IO_L3N_T0_DQS_EMCCLK_14
Sch=sw[1]
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { redSW[2] }]; #IO_L6N_T0_D08_VREF_14
Sch=sw[2]
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { redSW[3] }]; #IO_L13N_T2_MRCC_14
Sch=sw[3]
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { greenSW[0] }]; #IO_L12N_T1_MRCC_14
Sch=sw[4]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { greenSW[1] }]; #IO_L7N_T1_D10_14
Sch=sw[5]
```

```
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { greenSW[2] }]; #IO_L17N_T2_A13_D29_14
Sch=sw[6]
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports { greenSW[3] }]; #IO_L5N_T0_D07_14
Sch=sw[7]
set_property -dict { PACKAGE_PIN T8 IOSTANDARD LVCMOS18 } [get_ports { blueSW[0]}]; #IO_L24N_T3_34 Sch=sw[8]
set_property -dict { PACKAGE_PIN U8 IOSTANDARD LVCMOS18 } [get_ports { blueSW[1]}]; #IO_25_34 Sch=sw[9]
set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports { blueSW[2] }]; #IO_L15P_T2_DQS_RDWR_B_14
Sch=sw[10]
set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 } [get_ports { blueSW[3] }]; #IO_L23P_T3_A03_D19_14
Sch=sw[11]
set_property -dict { PACKAGE_PIN H6 IOSTANDARD LVCMOS33 } [get_ports { clkManagerSW[0] }]; #IO_L24P_T3_35
Sch=sw[12]
set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports { clkManagerSW[1] }];
#IO_L20P_T3_A08_D24_14 Sch=sw[13]
set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 } [get_ports { clkManagerSW[2] }];
#IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports { clkManagerSW[3] }]; #IO_L21P_T3_DQS_14
Sch=sw[15]


## RGB LEDs
set_property -dict { PACKAGE_PIN R12   IOSTANDARD LVCMOS33 } [get_ports { ledOutput[2] }]; #IO_L5P_T0_D06_14
Sch=led16_b
set_property -dict { PACKAGE_PIN M16   IOSTANDARD LVCMOS33 } [get_ports { ledOutput[1] }]; #IO_L10P_T1_D14_14
Sch=led16_g
set_property -dict { PACKAGE_PIN N15   IOSTANDARD LVCMOS33 } [get_ports { ledOutput[0] }]; #IO_L11P_T1_SRCC_14
Sch=led16_r
set_property -dict { PACKAGE_PIN G14   IOSTANDARD LVCMOS33 } [get_ports { ledOutput[5]}]; #IO_L15N_T2_DQS_ADV_B_15
Sch=led17_b
set_property -dict { PACKAGE_PIN R11   IOSTANDARD LVCMOS33 } [get_ports { ledOutput[4]}]; #IO_0_14 Sch=led17_g
set_property -dict { PACKAGE_PIN N16   IOSTANDARD LVCMOS33 } [get_ports { ledOutput[3]}]; #IO_L11N_T1_SRCC_14
Sch=led17_r


##Buttons
#set_property -dict { PACKAGE_PIN C12   IOSTANDARD LVCMOS33 } [get_ports { CPU_RESETN }]; #IO_L3P_T0_DQS_AD1P_15
Sch=cpu_resetn
set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports { rst }]; #IO_L9P_T1_DQS_14 Sch=btnc
set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 } [get_ports { switchProfileBtn }]; #IO_L4N_T0_D05_14
Sch=btnu
#set_property -dict { PACKAGE_PIN P17   IOSTANDARD LVCMOS33 } [get_ports { rstTop }]; #IO_L12P_T1_MRCC_14 Sch=btnl
#set_property -dict { PACKAGE_PIN M17   IOSTANDARD LVCMOS33 } [get_ports { ld }]; #IO_L10N_T1_D15_14 Sch=btnr
#set_property -dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33 } [get_ports { btn }]; #IO_L9N_T1_DQS_D13_14 Sch=btnd
```

Fifo.v

```
`timescale 1ns / 1ps
//`default_nettype  none

module fifo #(
    parameter WIDTH = 12,
    parameter DEPTH = 16
)(
    input [WIDTH-1:0] data_in,
    input wire clk,
    input wire write,
    input wire read,
    output reg [WIDTH-1:0] data_out,
    output wire fifo_full,
    output wire fifo_empty,
    output wire fifo_not_empty,
    output wire fifo_not_full
);
     integer count;
    // memory will contain the FIFO data.
    reg [WIDTH-1:0] memory [0:DEPTH-1];
    // $clog2(DEPTH+1)-2 to count from 0 to DEPTH
    reg [$clog2(DEPTH)-1:0] write_ptr;
    reg [$clog2(DEPTH)-1:0] read_ptr;


    assign fifo_empty   = ( write_ptr == read_ptr ) ? 1'b1 : 1'b0;
    assign fifo_full    = ( write_ptr == (DEPTH-1) ) ? 1'b1 : 1'b0;
    assign fifo_not_empty = ~fifo_empty;
    assign fifo_not_full = ~fifo_full;

    always @ (posedge clk) begin

        if ( write ) begin
            memory[write_ptr] <= data_in;
        end

        if ( read ) begin
            data_out <= memory[read_ptr];
        end

    end

    always @ ( posedge clk ) begin
        if ( write ) begin
            write_ptr <= write_ptr + 1;
        end

        if ( read && fifo_not_empty ) begin
            read_ptr <= read_ptr + 1;
        end
    end

endmodule
```

The fifo basically stores memory across a particular width at a certain depth. Write and read pointers are used to save the data into the memory array, and there is a counter for these pointers to store memory at different indexes. Flags of full and empty are used to prevent pushing data that doesn’t exist and prevent loading in too much data.

SERIALIZER.v

```
`timescale 1ns / 1ps


module SERIALIZER(
    input [11:0] pull_in,
    input pull_but,
    output reg [3:0] out_red,
    output reg [3:0] out_green,
    output reg [3:0] out_blue
    );
```

```
   reg state=0;

 always@(pull_but or pull_in)
     begin
        if (pull_but)
        state = ~state;
        else
        state = state;
     end

 always@(pull_but or state)
     begin
     case(state)
     1'b0:
     begin end
     1'b1:
     begin
        out_red <= pull_in[3:0];
        out_green <= pull_in[7:4];
        out_blue <= pull_in[11:8];
     end
        endcase
     end

endmodule

`timescale 1ns / 1ps


module SERIALIZER_TB(

    );
    reg [11:0] pull_in_tb;
    reg pull_but_tb;
    wire [3:0] out_red_tb;
    wire [3:0] out_green_tb;
    wire [3:0] out_blue_tb;

    SERIALIZER COMP(
    .pull_in(pull_in_tb),
    .pull_but(pull_but_tb),
    .out_red(out_red_tb),
    .out_green(out_green_tb),
    .out_blue(out_blue_tb)
    );

    initial
    begin
    pull_in_tb = 56;
    pull_but_tb = 1;
    #10
    pull_in_tb = 56;
    pull_but_tb = 0;
    #100
    $finish;
    end




endmodule
```

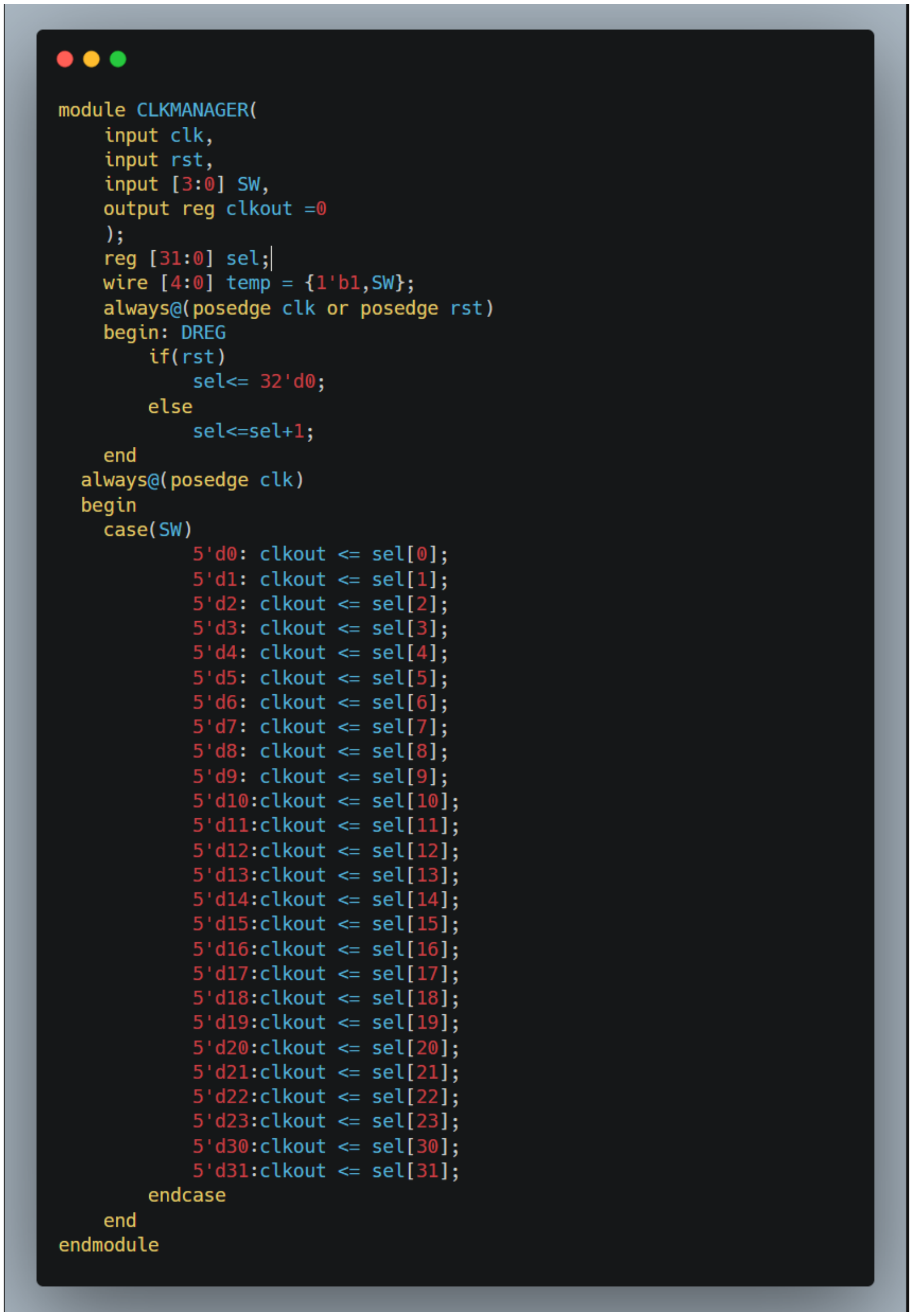
The serializer pulls an 12-bit number from the fifo using a pull button, and splits it into 3 4-bit numbers that go into the red, green, and blue PWM respectively.

Top.v

```
`timescale 1ns / 1ps


module top
(
    input[3:0] clkManagerSW,
    input[11:0] in,
    input clk,
    input rst,
    input writebut,
    input readbut,
    input pullbut,
    input switchProfileBtn,
    output full,
    output empty,
    output [6:0] Cnode1,
    output [7:0] AN1,
    output seg,
    output[5:0] ledOutput
    );
    wire clkMgrOut;
    CLKMANAGER clkManager(
    .clk(clk),
    .rst(rst),
    .SW(clkManagerSW),
    .clkout(clkMgrOut)
    );



    fifo fifo_wrapper(
   .data_in(in),
   .clk(clkMgrOut),
   .write(writebut),
   .read(readbut),
   .data_out(seg7Digit[23:12]),
   .fifo_full(full),
   .fifo_empty(empty)

    );

    wire [31:0] seg7Digit;
    SERIALIZER serial_wrapper (
    .pull_in(seg7Digit[23:12]),
    .pull_but(pullbut),
    .out_red(seg7Digit[3:0]),
    .out_green(seg7Digit[7:4]),
    .out_blue(seg7Digit[11:8])
    );

    wire tmpEn = 1'b1;
    wire [2:0] tmpLEDOUT;
    PWM redLED(
    .inputSW(seg7Digit[3:0]),
    .clk(clkMgrOut),
    .rst(rst),
    .en(tmpEn),
    .result(tmpLEDOUT[0])
    );

    PWM greenLED(
    .inputSW(seg7Digit[7:4]),
    .clk(clkMgrOut),
    .rst(rst),
    .en(tmpEn),
    .result(tmpLEDOUT[1])
    );

    PWM blueLED(
    .inputSW(seg7Digit[11:8]),
    .clk(clkMgrOut),
    .rst(rst),
    .en(tmpEn),
    .result(tmpLEDOUT[2])
    );

    FSM fsm(
    .sel(tmpLEDOUT),
    .fsmBTN(switchProfileBtn),
    .rst(rst),
    .clk(clkMgrOut),
    .led1(ledOutput[2:0]),
    .led2(ledOutput[5:3])
    );

    SEGDRIVE seg7driver_wrapper(
        .nexysCLK(clk),
        .inDigit(seg7Digit),
        .Cnode(Cnode1),
        .dp(seg),
        .AN(AN1)
    );

endmodule
```

Our top file instantiated the PWM redLed, PWM blueLed, and PWM greenLed, the fifo, serializer, and the segdriver. These three PWM modules output to the FSM module which then outputs to our XDC output. We also have a clock manager which is creating an artificial clock to input into each module. The fifo stores user input into a 2-D array then sends it off to a serializer which splits the data into 3 parts that each go into the input of a PWM.

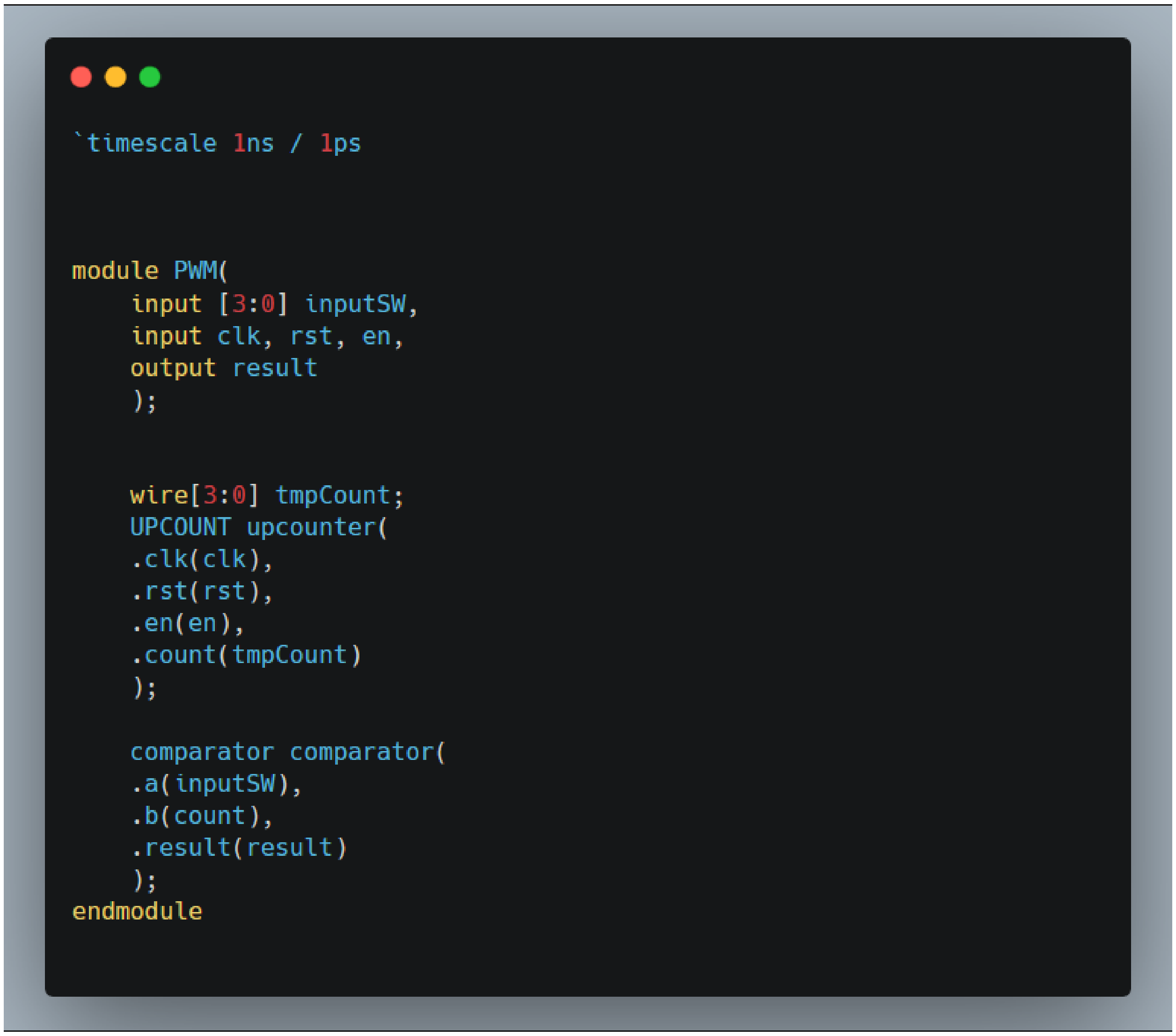
CLKMANAGER.v



```
module CLKMANAGER(
    input clk,
    input rst,
    input [3:0] SW,
    output reg clkout =0
    );
    reg [31:0] sel;
    wire [4:0] temp = {1'b1,SW};
    always@(posedge clk or posedge rst)
    begin: DREG
        if(rst)
            sel<= 32'd0;
        else
            sel<=sel+1;
    end
  always@(posedge clk)
  begin
    case(SW)
            5'd0: clkout <= sel[0];
            5'd1: clkout <= sel[1];
            5'd2: clkout <= sel[2];
            5'd3: clkout <= sel[3];
            5'd4: clkout <= sel[4];
            5'd5: clkout <= sel[5];
            5'd6: clkout <= sel[6];
            5'd7: clkout <= sel[7];
            5'd8: clkout <= sel[8];
            5'd9: clkout <= sel[9];
            5'd10:clkout <= sel[10];
            5'd11:clkout <= sel[11];
            5'd12:clkout <= sel[12];
            5'd13:clkout <= sel[13];
            5'd14:clkout <= sel[14];
            5'd15:clkout <= sel[15];
            5'd16:clkout <= sel[16];
            5'd17:clkout <= sel[17];
            5'd18:clkout <= sel[18];
            5'd19:clkout <= sel[19];
            5'd20:clkout <= sel[20];
            5'd21:clkout <= sel[21];
            5'd22:clkout <= sel[22];
            5'd23:clkout <= sel[23];
            5'd30:clkout <= sel[30];
            5'd31:clkout <= sel[31];
        endcase
    end
endmodule
```

We used a 32 bit clock manager to feed variable frequencies into our PWM LED modules, and our FSM module.

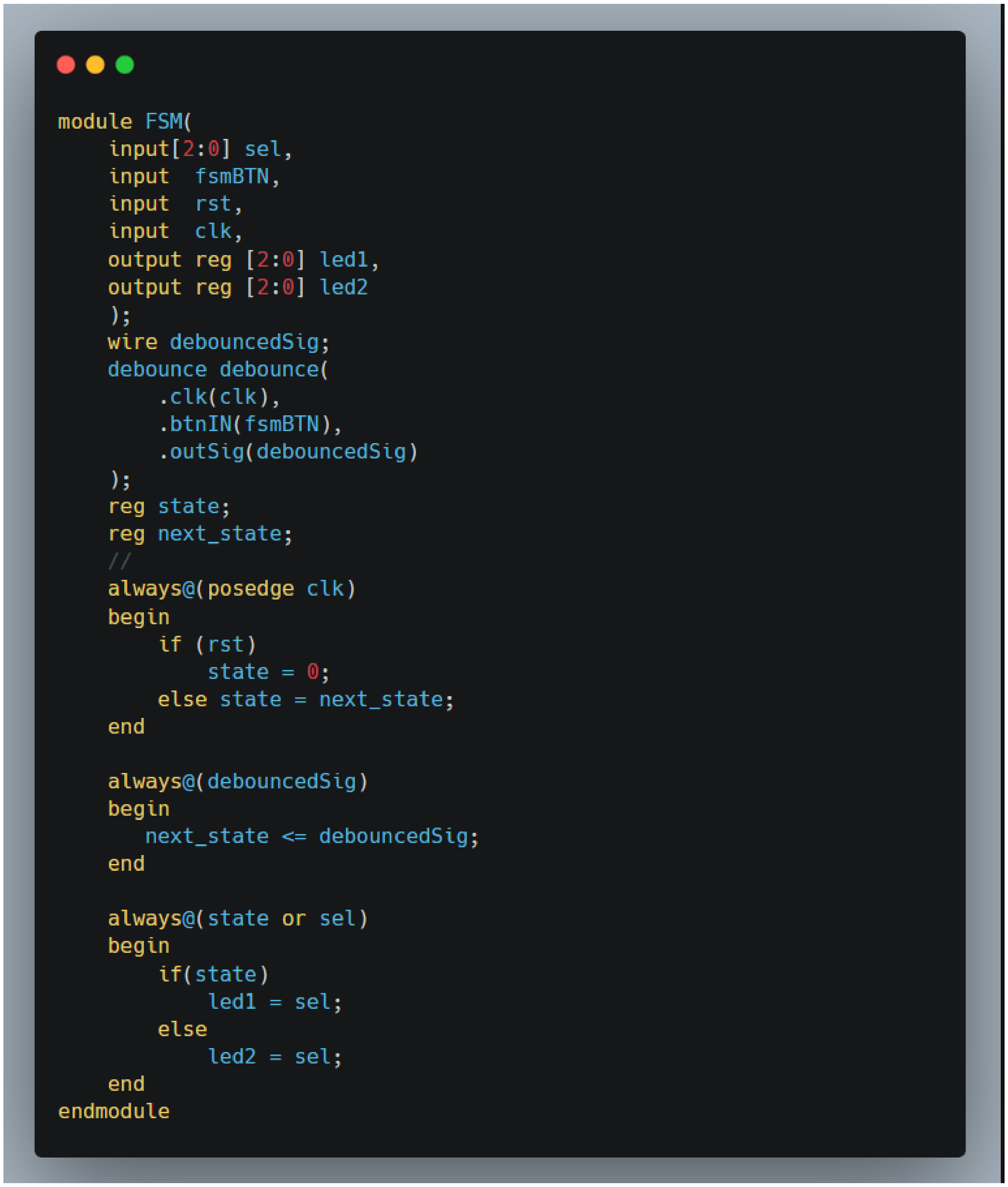
PWM.v



```
`timescale 1ns / 1ps



module PWM(
    input [3:0] inputSW,
    input clk, rst, en,
    output result
    );


    wire[3:0] tmpCount;
    UPCOUNT upcounter(
    .clk(clk),
    .rst(rst),
    .en(en),
    .count(tmpCount)
    );

    comparator comparator(
    .a(inputSW),
    .b(count),
    .result(result)
    );
endmodule
```

This is the generic PWM file which instantiates into the redLed, blueLed, and greenLed to send off those values to the FSM and eventually to the dedicated output led on our Nexys A7 board. It uses the UPCOUNT and comparator modules to compare the values from the switches to the counter and based off those values it will send the appropriate signals to the FSM for outputting.

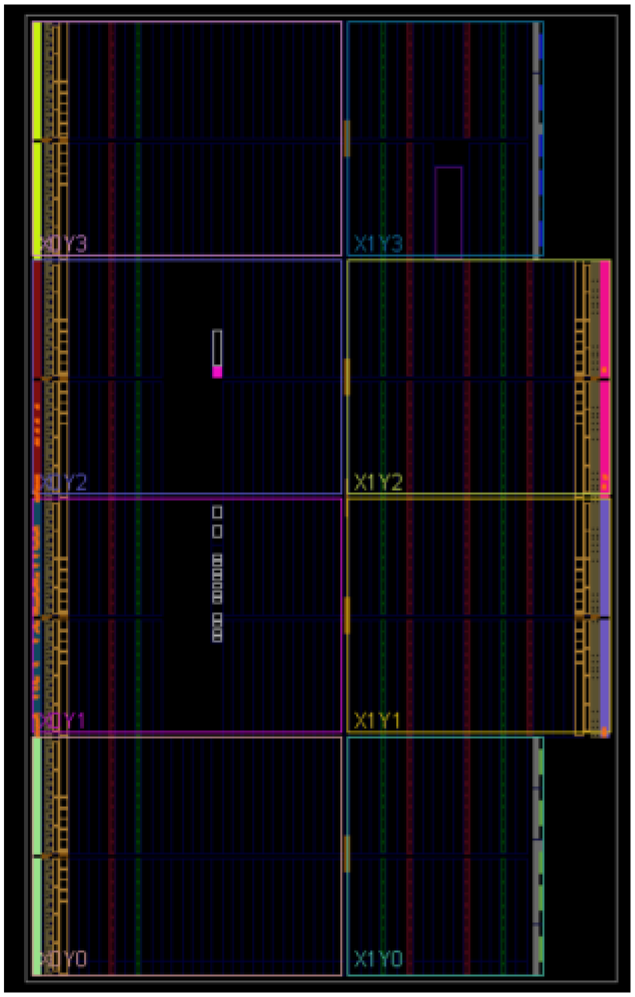
FSM.v

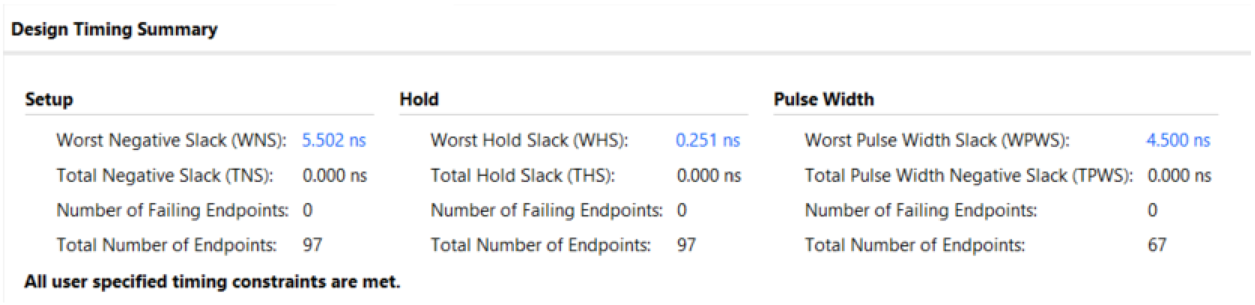


```
module FSM(
    input[2:0] sel,
    input  fsmBTN,
    input  rst,
    input  clk,
    output reg [2:0] led1,
    output reg [2:0] led2
    );
    wire debouncedSig;
    debounce debounce(
        .clk(clk),
        .btnIN(fsmBTN),
        .outSig(debouncedSig)
    );
    reg state;
    reg next_state;
    //
    always@(posedge clk)
    begin
        if (rst)
            state = 0;
        else state = next_state;
    end

    always@(debouncedSig)
    begin
       next_state <= debouncedSig;
    end

    always@(state or sel)
    begin
        if(state)
            led1 = sel;
        else
            led2 = sel;
    end
endmodule
```

This simple FSM (finite state machine) module will iterate between two states depending on the input button. Depending on the button output (which is debounced using the button debouncer) it will either fill the led1 values or led2 values.

IMPLEMENTED DESIGN/ TIMING SUMMARY:





Design Timing Summary

Setup

```
Worst Negative Slack (WNS):    5.502 ns
Total Negative Slack (TNS):    0.000 ns
Number of Failing Endpoints:   0
Total Number of Endpoints:     97
```

Hold

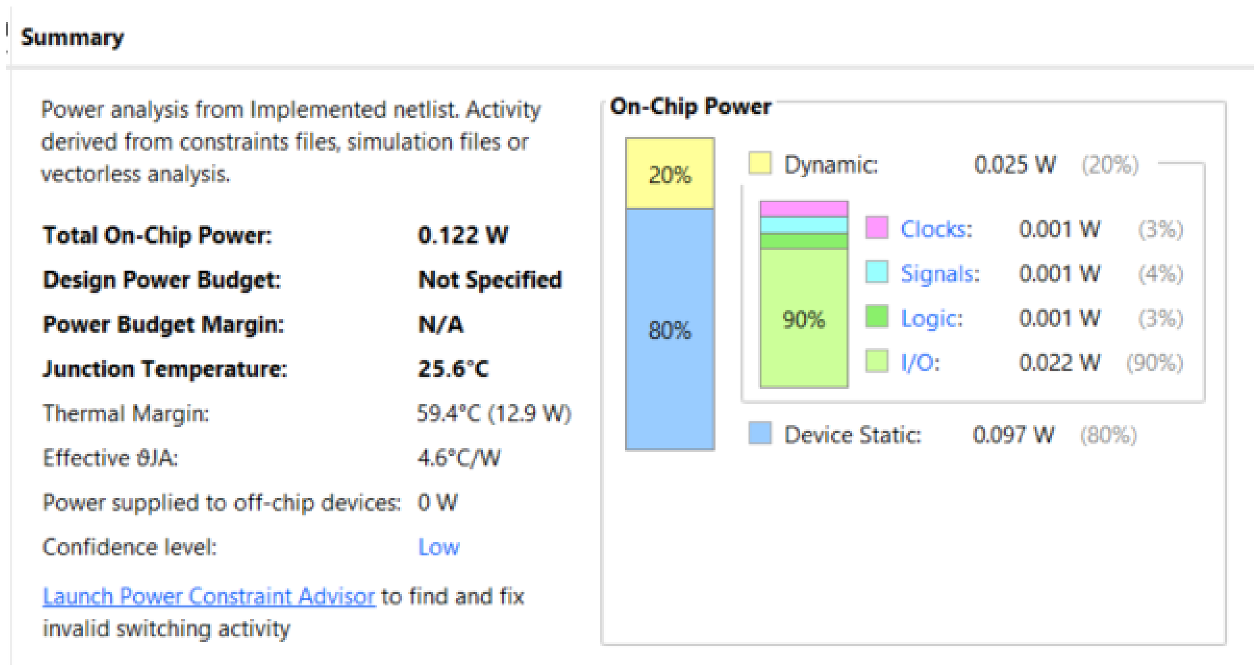
```
Worst Hold Slack (WHS):        0.251 ns
Total Hold Slack (THS):        0.000 ns
Number of Failing Endpoints:   0
Total Number of Endpoints:     97
```

Pulse Width

```
Worst Pulse Width Slack (WPWS):                4.500 ns
Total Pulse Width Negative Slack (TPWS):       0.000 ns
Number of Failing Endpoints:                   0
Total Number of Endpoints:                     67
```

All user specified timing constraints are met.

POWER SUMMARY:



Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

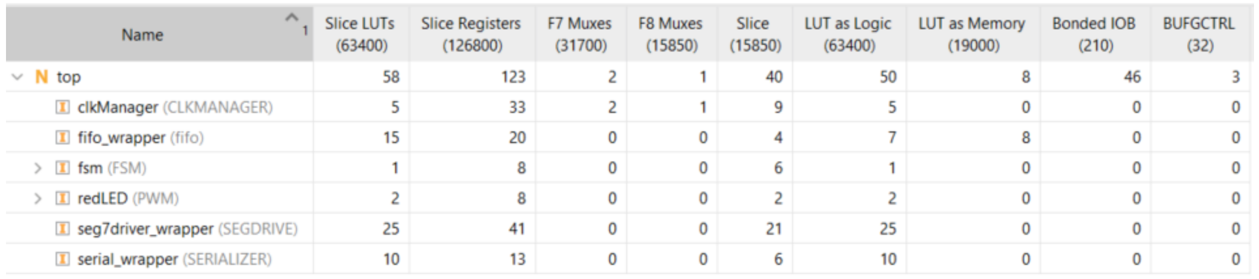
Total On-Chip Power:	**0.122 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25.6°C**
Thermal Margin:	59.4°C (12.9 W)
Effective ϑJA:	4.6°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

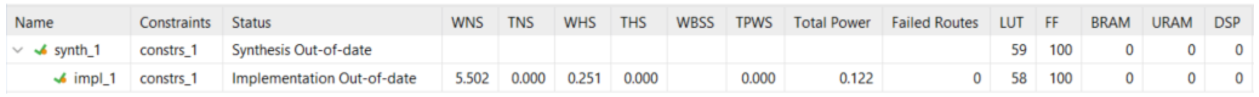
🟨	Dynamic:	0.025 W	(20%)
🟪	Clocks:	0.001 W	(3%)
🟦	Signals:	0.001 W	(4%)
🟩	Logic:	0.001 W	(3%)
🟩	I/O:	0.022 W	(90%)
🟦	Device Static:	0.097 W	(80%)

UTILIZATION:



Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	F8 Muxes (15850)	Slice (15850)	LUT as Logic (63400)	LUT as Memory (19000)	Bonded IOB (210)	BUFGCTRL (32)
top	58	123	2	1	40	50	8	46	3
clkManager (CLKMANAGER)	5	33	2	1	9	5	0	0	0
fifo_wrapper (fifo)	15	20	0	0	4	7	8	0	0
fsm (FSM)	1	8	0	0	6	1	0	0	0
redLED (PWM)	2	8	0	0	2	2	0	0	0
seg7driver_wrapper (SEGDRIVE)	25	41	0	0	21	25	0	0	0
serial_wrapper (SERIALIZER)	10	13	0	0	6	10	0	0	0

RESOURCE USAGE:



Name	Constraints	Status	WNS	TNS	WHS	THS	WBSS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP
synth_1	constrs_1	Synthesis Out-of-date									59	100	0	0	0
impl_1	constrs_1	Implementation Out-of-date	5.502	0.000	0.251	0.000		0.000	0.122	0	58	100	0	0	0